### [Micropython][ESP8266]TPYBoard V202之控制OLED液晶显示屏

1.实验目的

1.学习在PC机系统中扩展简单I/O 接口的方法。

2.进一步学习编制数据输出程序的设计方法。

3.学习TPYBoard v202控制OLED显示字符。

2.所需元器件

TPYBoard v202开发板一块

数据线一条

杜邦线若干

OLED液晶屏一块

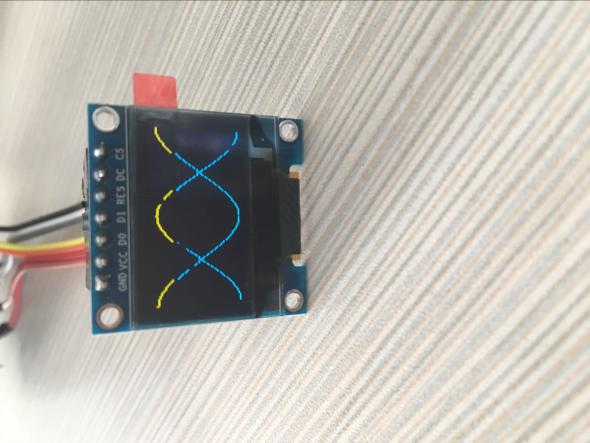
3.什么是OLED显示屏

(1) OLED显示屏简介

有机发光二极管（organic light-emitting diode，OLED）是一种由柯达公司开发并拥有专利的显示技术，这项技术使用有机聚合材料作为发光二极管中的半导体（semiconductor）材料。聚合材料可以是天然的，也可能是人工合成的，可能尺寸很大，也可能尺寸很小。其广泛运用于手机、数码摄像机、DVD机、个人数字助理（PDA）、笔记本电脑、汽车音响和电视。OLED显示器很薄很轻，因为它不使用背光。

本例中使用0.96 寸OLED显示屏，该屏具有高亮度，低功耗屏，显示颜色纯正，在阳光下有很好的可视效果。模块供电可以是3.3V 也可以是5V，不需要修改模块电路，同时兼容3种通信方式：4 线SPI、3线SPI、 IIC，通信模式的选择可以根据提供的BOM表进行跳选。该模块一共有三种颜色：蓝色、白色、黄蓝双色。OLED 屏具有多个控制指令，可以控制OLED 的亮度、对比度、开关升压电路等指令。操作方便，功能丰富。同时为了方便应用在产品上，预留4个M2 固定孔，方便用户固定在机壳上。0.96寸OLED显示屏的驱动芯片为：SSD1306(已集成在屏中)。

(2)实际显示效果



(3)OLED接口定义

1> GND= 电源地

2> VCC= 电源地（2.8V~5.5V）

3> D0 = 时钟线

4> D1 = 数据线

5> RES= 复位线

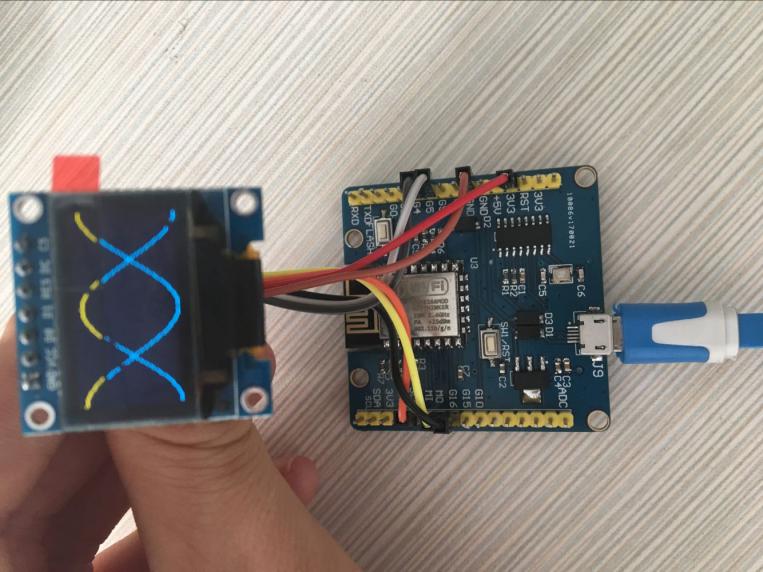
6> DC = 数据/命令

7> CS = 片选

1. 具体接线方法

|  |  |
| --- | --- |
| TPYboard v202 | OLED |
| GND | GND |
| 3.3V | VCC |
| SCK | D0 |
| MO | D1 |
| G4 | RES |
| G5 | DC |
| G16 | CS |

1. 实物接线图



1. 程序源代码

main.py 程序源代码

# main.py -- put your code here!

import machine

from machine import Pin,I2C,SPI

import ssd1306

import math

import time

spi = SPI(baudrate=10000000, polarity=1, phase=0, sck=Pin(14,Pin.OUT), mosi=Pin(13,Pin.OUT), miso=Pin(12))

display = ssd1306.SSD1306\_SPI(128, 64, spi, Pin(5),Pin(4), Pin(16))

led\_blue = machine.Pin(2, Pin.OUT) # 设置 GPIO2 为输出

led\_blue.high()

try:

display.poweron()

display.init\_display()

display.text('TPYBoard V202',1,1)

display.text('Hi, TurnipSmart',1,16)

display.text('I Love You',1,31)

display.text('This is DNA!!',1,46)

display.show()

time.sleep(3)

display.fill(0)

#显示DNA

for x in range(0, 128):

display.pixel(x, 32+int(math.cos(x/64\*math.pi)\*30 +2), 1)

display.pixel(x, 32+int(math.cos((x+64)/64\*math.pi)\*30+2), 1)

display.show()

except Exception as ex:

led\_blue.low()

print('Unexpected error: {0}'.format(ex))

display.poweroff()

ssd1306程序源代码

import pyb

import font

# Constants

DISPLAYOFF = 0xAE

SETCONTRAST = 0x81

DISPLAYALLON\_RESUME = 0xA4

DISPLAYALLON = 0xA5

NORMALDISPLAY = 0xA6

INVERTDISPLAY = 0xA7

DISPLAYON = 0xAF

SETDISPLAYOFFSET = 0xD3

SETCOMPINS = 0xDA

SETVCOMDETECT = 0xDB

SETDISPLAYCLOCKDIV = 0xD5

SETPRECHARGE = 0xD9

SETMULTIPLEX = 0xA8

SETLOWCOLUMN = 0x00

SETHIGHCOLUMN = 0x10

SETSTARTLINE = 0x40

MEMORYMODE = 0x20

COLUMNADDR = 0x21

PAGEADDR = 0x22

COMSCANINC = 0xC0

COMSCANDEC = 0xC8

SEGREMAP = 0xA0

CHARGEPUMP = 0x8D

EXTERNALVCC = 0x10

SWITCHCAPVCC = 0x20

SETPAGEADDR = 0xB0

SETCOLADDR\_LOW = 0x00

SETCOLADDR\_HIGH = 0x10

ACTIVATE\_SCROLL = 0x2F

DEACTIVATE\_SCROLL = 0x2E

SET\_VERTICAL\_SCROLL\_AREA = 0xA3

RIGHT\_HORIZONTAL\_SCROLL = 0x26

LEFT\_HORIZONTAL\_SCROLL = 0x27

VERTICAL\_AND\_RIGHT\_HORIZONTAL\_SCROLL = 0x29

VERTICAL\_AND\_LEFT\_HORIZONTAL\_SCROLL = 0x2A

# I2C devices are accessed through a Device ID. This is a 7-bit

# value but is sometimes expressed left-shifted by 1 as an 8-bit value.

# A pin on SSD1306 allows it to respond to ID 0x3C or 0x3D. The board

# I bought from ebay used a 0-ohm resistor to select between "0x78"

# (0x3c << 1) or "0x7a" (0x3d << 1). The default was set to "0x78"

DEVID = 0x3c

# I2C communication here is either <DEVID> <CTL\_CMD> <command byte>

# or <DEVID> <CTL\_DAT> <display buffer bytes> <> <> <> <>...

# These two values encode the Co (Continuation) bit as b7 and the

# D/C# (Data/Command Selection) bit as b6.

CTL\_CMD = 0x80

CTL\_DAT = 0x40

class SSD1306(object):

def \_\_init\_\_(self, pinout, height=32, external\_vcc=True, i2c\_devid=DEVID):

self.external\_vcc = external\_vcc

self.height = 32 if height == 32 else 64

self.pages = int(self.height / 8)

self.columns = 128

# Infer interface type from entries in pinout{}

if 'dc' in pinout:

# SPI

rate = 16 \* 1024 \* 1024

self.spi = pyb.SPI(1, pyb.SPI.MASTER, baudrate=rate, polarity=1, phase=0) # SCK: Y6: MOSI: Y8

self.dc = pyb.Pin(pinout['dc'], pyb.Pin.OUT\_PP, pyb.Pin.PULL\_DOWN)

self.res = pyb.Pin(pinout['res'], pyb.Pin.OUT\_PP, pyb.Pin.PULL\_DOWN)

self.offset = 0

else:

# Infer bus number from pin

if pinout['sda'] == 'X10':

self.i2c = pyb.I2C(1)

else:

self.i2c = pyb.I2C(2)

self.i2c.init(pyb.I2C.MASTER, baudrate=400000) # 400kHz

self.devid = i2c\_devid

# used to reserve an extra byte in the image buffer AND as a way to

# infer the interface type

self.offset = 1

# I2C command buffer

self.cbuffer = bytearray(2)

self.cbuffer[0] = CTL\_CMD

def clear(self):

self.buffer = bytearray(self.offset + self.pages \* self.columns)

if self.offset == 1:

self.buffer[0] = CTL\_DAT

def write\_command(self, command\_byte):

if self.offset == 1:

self.cbuffer[1] = command\_byte

self.i2c.send(self.cbuffer, addr=self.devid, timeout=5000)

else:

self.dc.low()

self.spi.send(command\_byte)

def invert\_display(self, invert):

self.write\_command(INVERTDISPLAY if invert else NORMALDISPLAY)

def display(self):

self.write\_command(COLUMNADDR)

self.write\_command(0)

self.write\_command(self.columns - 1)

self.write\_command(PAGEADDR)

self.write\_command(0)

self.write\_command(self.pages - 1)

if self.offset == 1:

self.i2c.send(self.buffer, addr=self.devid, timeout=5000)

else:

self.dc.high()

self.spi.send(self.buffer)

def set\_pixel(self, x, y, state):

index = x + (int(y / 8) \* self.columns)

if state:

self.buffer[self.offset + index] |= (1 << (y & 7))

else:

self.buffer[self.offset + index] &= ~(1 << (y & 7))

def init\_display(self):

chargepump = 0x10 if self.external\_vcc else 0x14

precharge = 0x22 if self.external\_vcc else 0xf1

multiplex = 0x1f if self.height == 32 else 0x3f

compins = 0x02 if self.height == 32 else 0x12

contrast = 0xff # 0x8f if self.height == 32 else (0x9f if self.external\_vcc else 0x9f)

data = [DISPLAYOFF,

SETDISPLAYCLOCKDIV, 0x80,

SETMULTIPLEX, multiplex,

SETDISPLAYOFFSET, 0x00,

SETSTARTLINE | 0x00,

CHARGEPUMP, chargepump,

MEMORYMODE, 0x00,

SEGREMAP | 0x10,

COMSCANDEC,

SETCOMPINS, compins,

SETCONTRAST, contrast,

SETPRECHARGE, precharge,

SETVCOMDETECT, 0x40,

DISPLAYALLON\_RESUME,

NORMALDISPLAY,

DISPLAYON]

for item in data:

self.write\_command(item)

self.clear()

self.display()

def poweron(self):

if self.offset == 1:

pyb.delay(10)

else:

self.res.high()

pyb.delay(1)

self.res.low()

pyb.delay(10)

self.res.high()

pyb.delay(10)

def poweroff(self):

self.write\_command(DISPLAYOFF)

def contrast(self, contrast):

self.write\_command(SETCONTRAST)

self.write\_command(contrast)

def draw\_text(self, x, y, string, size=1, space=1):

def pixel\_x(char\_number, char\_column, point\_row):

char\_offset = x + char\_number \* size \* font.cols + space \* char\_number

pixel\_offset = char\_offset + char\_column \* size + point\_row

return self.columns - pixel\_offset

def pixel\_y(char\_row, point\_column):

char\_offset = y + char\_row \* size

return char\_offset + point\_column

def pixel\_mask(char, char\_column, char\_row):

char\_index\_offset = ord(char) \* font.cols

return font.bytes[char\_index\_offset + char\_column] >> char\_row & 0x1

pixels = (

(pixel\_x(char\_number, char\_column, point\_row),

pixel\_y(char\_row, point\_column),

pixel\_mask(char, char\_column, char\_row))

for char\_number, char in enumerate(string)

for char\_column in range(font.cols)

for char\_row in range(font.rows)

for point\_column in range(size)

for point\_row in range(1, size + 1))

for pixel in pixels:

self.set\_pixel(\*pixel)